

This Page Is Inserted by IFW Operations  
and is not a part of the Official Record

## **BEST AVAILABLE IMAGES**

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

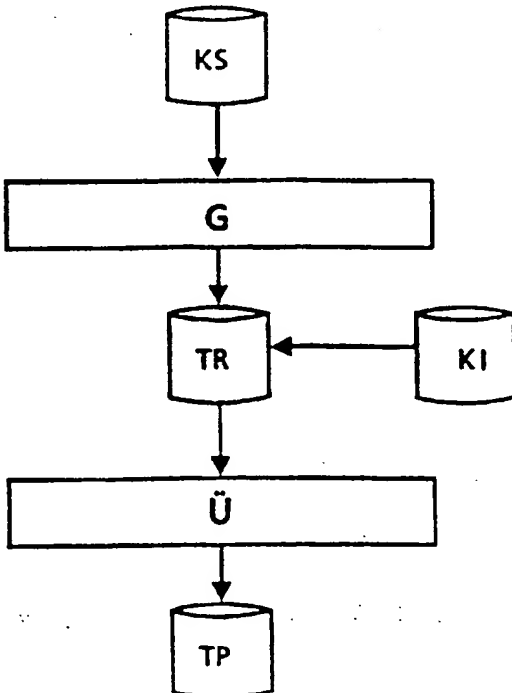
**As rescanning documents *will not* correct images,  
please do not report the images to the  
Image Problems Mailbox.**



**THIS PAGE BLANK (USPTO)**



WELTORGANISATION FÜR GEISTIGES EIGENTUM  
Internationales Büro

<p>(51) Internationale Patentklassifikation 5 : <b>G06F 11/00</b></p>	<p><b>A1</b></p>	<p>(11) Internationale Veröffentlichungsnummer: <b>WO 94/14117</b></p> <p>(43) Internationales Veröffentlichungsdatum: <b>23. Juni 1994 (23.06.94)</b></p>
<p>(21) Internationales Aktenzeichen: <b>PCT/EP93/03450</b></p> <p>(22) Internationales Anmeldedatum: <b>8. Dezember 1993 (08.12.93)</b></p> <p>(30) Prioritätsdaten: 92121511.7      17. Dezember 1992 (17.12.92)      EP (34) Länder für die die regionale oder internationale Anmeldung eingereicht worden ist: <b>DE usw.</b></p> <p>(71) Anmelder (für alle Bestimmungsstaaten ausser US): <b>SIEMENS AKTIENGESELLSCHAFT [DE/DE]; Wittelsbacherplatz 2, D-80333 München (DE).</b></p> <p>(72) Erfinder; und (75) Erfinder/Anmelder (nur für US): <b>KOLB, Sebald [DE/DE]; Clemens-Schöps-Strasse 2a, D-85521 Ottobrunn (DE).</b></p>		<p>(81) Bestimmungsstaaten: <b>JP, US, europäisches Patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).</b></p> <p><b>Veröffentlicht</b> <i>Mit internationalem Recherchenbericht. Vor Ablauf der für Änderungen der Ansprüche zugelassenen Frist. Veröffentlichung wird wiederholt falls Änderungen eintreffen.</i></p>
<p>(54) Title: <b>PROCESS FOR TESTING AT LEAST ONE CLASS OF AN OBJECT-ORIENTED PROGRAM ON A COMPUTER</b></p> <p>(54) Bezeichnung: <b>VERFAHREN ZUM TESTEN MINDESTENS EINER KLASSE EINES OBJEKTORIENTIERTEN PROGRAMMES AUF EINEM RECHNER</b></p> <p>(57) Abstract</p> <p>The invention provides for the first time a process for testing classes of an object-oriented program. Classes are tested by causing the tests interactively to input test commands with their method calls. It is also possible to check the test results interactively. Testability is obtained in that the necessary call parameters are found beforehand for all the methods permitted in the class and precautions are taken in the computer store to store and alter the possible parameters. The advantage is that testing can be performed interactively and not, as previously, in that the test program had to be once more corrected, translated and fixed after test evaluation, which took a great deal of time. The invention can advantageously be used in the development of communication software.</p> <p>(57) Zusammenfassung</p> <p>Mit der Erfindung wird erstmals ein Verfahren zum Testen von Klassen eines objektorientierten Programmes zur Verfügung gestellt. Klassen werden getestet, indem vom Tester Testkommandos mit denen Methodenaufrufe möglich sind, interaktiv eingegeben werden. die Überprüfung der Testergebnisse ist ebenfalls interaktiv möglich. Erreicht wird die Testbarkeit dadurch, daß vorab für alle in der Klasse erlaubten Methoden, die erforderlichen Aufrufparameter ermittelt werden, und daß im Rechnerspeicher Vorkehrungen getroffen werden, um die möglichen Parameter abzuspeichern und zu verändern. Der Vorteil besteht darin, daß interaktiv getestet werden kann, und nicht, wie bisher üblich das Testprogramm nach der Testauswertung zeitaufwendig erneut korrigiert übersetzt und gebunden werden muß. Vorteilhaft kann die Erfindung bei der Entwicklung von Kommunikations-Software eingesetzt werden.</p>		
 <pre> graph TD     KS[(KS)] --&gt; G[G]     G --&gt; TR[(TR)]     KI[(KI)] --&gt; TR     TR --&gt; U[Ü]     U --&gt; TP[(TP)] </pre>		



### LEDIGLICH ZUR INFORMATION

Codes zur Identifizierung von PCT-Vertragsstaaten auf den Kopfbögen der Schriften, die internationale Anmeldungen gemäss dem PCT veröffentlichen.

AT	Österreich	GA	Gabon	MR	Mauretanien
AU	Australien	GB	Vereinigtes Königreich	MW	Malawi
BB	Barbados	GE	Georgien	NE	Niger
BE	Belgien	GN	Guinea	NL	Niederlande
BF	Burkina Faso	GR	Griechenland	NO	Norwegen
BG	Bulgarien	HU	Ungarn	NZ	Neuseeland
BJ	Benin	IE	Irland	PL	Polen
BR	Brasilien	IT	Italien	PT	Portugal
BY	Belarus	JP	Japan	RO	Rumänien
CA	Kanada	KE	Kenya	RU	Russische Föderation
CF	Zentrale Afrikanische Republik	KG	Kirgisistan	SD	Sudan
CG	Kongo	KP	Demokratische Volksrepublik Korea	SE	Schweden
CH	Schweiz	KR	Republik Korea	SI	Slowenien
CI	Côte d'Ivoire	KZ	Kasachstan	SK	Slowakei
CM	Kamerun	LI	Liechtenstein	SN	Senegal
CN	China	LK	Sri Lanka	TD	Tschad
CS	Tschechoslowakei	LU	Luxemburg	TG	Togo
CZ	Tschechische Republik	LV	Lettland	TJ	Tadschikistan
DE	Deutschland	MC	Monaco	TT	Trinidad und Tobago
DK	Dänemark	MD	Republik Moldau	UA	Ukraine
ES	Spanien	MG	Madagaskar	US	Vereinigte Staaten von Amerika
FI	Finnland	ML	Mali	UZ	Usbekistan
FR	Frankreich	MN	Mongolei	VN	Vietnam



Verfahren zum Testen mindestens einer Klasse eines objektorientierten Programmes auf einem Rechner

Mit dem zunehmendem Umfang von Systemlösungen im Rahmen von industriellen Großprojekten, wächst auch der Beitrag den die Software zur Lösung liefert. Damit man große Softwareprojekte effizient abwickeln kann, hat es sich als vorteilhaft erwiesen, diese Projekte in Teilprojekte zu unterteilen und einzelne Programme modular zu erstellen. Ein nicht unwesentlicher Aspekt bei der modularen Erstellung von Programmen ist auch, daß einzelne Programmmoduln von anderen Projekten leicht wieder verwertet werden können. In den vergangenen Jahren hat sich in diesem Zusammenhang besonders die Technik des objektorientierten Programmierens als vorteilhaft erwiesen. Eine Einführung in die Grundlagen und die Namensgebung der objektorientierten Programmierung findet sich zum Beispiel in [1].

Im Zusammenhang mit der Wiederverwertbarkeit einzelner Programmmoduln, kommt besonders der Qualitätssicherung dieser Programmmoduln eine erhöhte Bedeutung zu. Den Anforderungen einer besonders fehlerfreien Qualität einzelner Programmstrukturen wird durch besonders entwickelte Testverfahren Rechnung getragen. Ein wesentliches Anliegen des Softwaretestens ist es, die Qualität der Software im Rahmen der Testabdeckung nachzuweisen. Um den Tester bei seiner Arbeit zu unterstützen, sind unterschiedlichste Testverfahren denkbar. Man will mit diesen Testverfahren einen durchgehenden Test der Software von einzelnen Programmmoduln bis zum kompletten System ermöglichen. Insbesondere unterscheidet man dabei drei aufeinander aufbauende Tests: den Komponententest, den Integrationstest und den Systemtest. Der Komponententest hat das Ziel, Fehler in einzelnen Programmkomponenten zu finden. Bei den Komponenten handelt es sich herkömmlicherweise um ein oder mehrere Moduln, beispielsweise in der objektorientierten Programmierung um Klassen. Der Komponententest wird durch den



Integrationstest, mit dessen Hilfe Fehler an den Schnittstellen und in der Kommunikation zwischen den getesteten Komponenten

entdeckt werden, ergänzt. Zum Abschluß kann ein Systemtest durchgeführt werden, welcher einen Test aus Kundensicht darstellt (z.B. der Test einer kompletten vermittlungstechnischen Anlage). Er hat das Ziel die Stabilität des gesamten (Programm-)Systems unter "Last", d.h. realen bis extremen Bedingungen, zu testen.

Im Rahmen der Technik des objektorientierten Programmierens wird der Komponententest durch ein Testverfahren für Klassen bereitgestellt. Sinnvollerweise beginnt man mit dem Testen bei einzelnen Programmkomponenten, d.h. den Klassen. Setzt man beim Test eines Programmes auf nicht ausgetestete Klassen auf, so werden Fehler, die bei der Klassenimplementierung gemacht wurden, nicht oder nur sehr viel schwerer gefunden und entsprechend schwer behoben. Der Grund dafür liegt neben der schlechten Überschaubarkeit des gesamten Programmes in der Vermischung dieser funktionalen Fehler, mit denen, die beim Zusammenspiel der einzelnen Objekte auftreten (Kommunikations-bzw. Schnittstellenfehlern). Ein weiterer wesentlicher Grund für gut ausgetestete Klassen ist die daraus resultierende gesteigerte Bereitschaft, diese Klassen wieder zu verwenden. Qualitativ schlechte Software eignet sich offensichtlich nicht zur Wiederverwendung.

Klassen sind als Testlinge in der Regel für sich alleine nicht ablauffähig und damit ohne zusätzlichen Aufwand auch nicht mit einem Debugger eines Betriebssystems zu debuggen. Da diese Testlingsklassen nur Programmstrukturen darstellen, ist es zwingend erforderlich, für ihren Test Objekte aus ihnen abzuleiten. Sie werden dann getestet, indem bei den Objekten entsprechende Methoden aufgerufen und ausgeführt werden. Die Parameter eines Methodenaufrufs zusammen mit den Werten der



Datenkomponenten eines aufzurufenden Objekts stellen dann die Testdaten dar.

Die Technik des objektorientierten Programmierens ist vergleichsweise jung. Es gibt deshalb keine technischen Verfahren zum testen von Klassen objektorientierter Software. Würde ein Programmierer auf herkömmliche Weise eine Klasse testen wollen, so müßte er zunächst einen Testfall generieren, d.h. ein Programm schreiben, welches aus einer Klasse ein Objekt instanziert. Weiterhin müßte er im Vorfeld des Tests Methodenaufrufe inklusive Parameterwerte für dieses Objekt in dieses Programm einarbeiten, welche möglichst genau auf die Fehler, die er zu finden hofft, abgestimmt wären. Anschließend würde er dieses Programm übersetzen und binden müssen. Nach dem Testlauf würde dann eine Auswertung erfolgen. Falls der Test keine Aussagen über die Funktionsfähigkeit dieser Klasse zuließe, müßte er den gesamten Vorgang erneut durchführen.

Die der Erfindung zugrundeliegende Aufgabe besteht darin, ein interaktives Verfahren zum Testen mindestens einer Klasse eines objektorientierten Programmes auf einem Rechner anzugeben.

Diese Aufgabe wird gemäß den Merkmalen des Patentanspruchs 1 gelöst.

Alle übrigen Weiterbildungen der Erfindung ergeben sich aus den Unteransprüchen.

Besonders vorteilhaft an dem angegebenen Verfahren ist die Tatsache, daß Klassen zeitlich vor und unabhängig von ihrer eigentlichen Anwendung in objektorientierten Programmen getestet werden können.



Besonders vorteilhaft an der Anwendung des erfindungsgemäßen Verfahrens ist die Tatsache, daß bereits im Vorfeld des Tests alle für den Test maßgeblichen Kenngrößen erfaßt werden und daß das Testprogramm nur einmal übersetzt und gebunden werden muß.

Um das Zusammenspiel auch mehrerer Klassen sinnvoll testen zu können, ist im erfindungsgemäßen Verfahren günstigerweise vorgesehen, die Methodeninformation aller am Test beteiligten Klassen für die in der externen Methodenschnittstelle der beteiligten Klassen definierten Methoden zu gewinnen.

Besonders vorteilhaft ist am erfindungsgemäßen Verfahren, daß es im Vorfeld eine Speicherzuweisung für die Objektinitialisierung oder Methodenaufrufe vorsieht.

Eine günstige Form der Speicherzuweisung geschieht beispielsweise über pointer, die dann während des Testbetriebs undefiniert werden können.

Weiterhin ist es günstig, daß das erfindungsgemäße Verfahren einen Zugriff auf objektinterne Datenkomponenten vorsieht, da diese normalerweise für einen Tester wegen des, beim objektorientierten Programmieren üblichen, Data Hidings nicht zugänglich sind.

Weiterhin ist es beim angegebenen Verfahren günstig, daß ein Debugger angeschlossen ist, sodaß beim Test entdeckte Fehler sofort mit zusätzlicher Zuhilfenahme des Debuggers analysiert und lokalisiert werden können.

Um die Funktionsweise einer Klasse möglichst vollständig Testen zu können, sieht das erfindungsgemäße Verfahren sinnvollerweise auch eine Gewinnung der Methodeninformation für generische und vererbende Klassen vor.



Im folgenden wird die Erfindung anhand von Figuren und Beispielen weiter erläutert.

Figur 1 zeigt ein Ausführungsbeispiel eines erfindungsgemäßen Verfahrens. Dieses Ausführungsbeispiel kann sich beispielsweise auf eine Vermittlungseinrichtung beziehen. Die Programmiersprache, in der der Rechner programmiert wird, sei beispielsweise Object CHILL ([2]).

Figur 2 zeigt als Beispiel ein Programmlisting

Figur 3 zeigt ein Blockdiagramm eines erfindungsgemäßen Verfahrens

Wie in Figur 1 dargestellt, wird beispielsweise aus einer Klassenspezifikation KS mit Hilfe eines Generators G Methodeninformation gewonnen und diese Methodeninformation dazu benutzt, Variablenspeicher für die entsprechenden Parameter der Methodenaufrufe und die Objekte der Testlingsklassen selbst in Form eines Testrahmens TR bereitzustellen. Anschließend wird mit Hilfe eines Übersetzers Ü der Testrahmen TR zusammen mit einem Kommandointerpreter KI übersetzt, wodurch ein fertiges Testprogramm TP entsteht. Dieser Kommandointerpreter KI ist beispielsweise für eine Testkommandosprache (die Befehlssyntax ist weiter hinten in der Beschreibung unter "Eingaben" erläutert) entworfen. So kann der Testende einfach Testszenarien erstellen. Beispielsweise kann der Kommandointerpreter folgende Befehle zulassen:

```
<statements> ::= {<command> ";"}.  
<command> ::= <control_cmd> | <general_cmd> | <regtest_cmd> |  
               <output_cmd> | <modclss_cmd> | <assign_cmd> |  
               <methcall_cmd>
```



```

<control_cmd> ::= Do WHILE <chars> ";" <statements> OD
                IF <chars> THEN <statements>
                [ELSE <statements>] FI.

<general_cmd> ::= DCL <chars> |
                  END |
                  PROTSTATUS |
                  PROTOCOL [TERM] [ALL] [INP] [OUT] |
                  INPUT (INP1 | INP2 | INP3 | INP4 | INP5) |
                  RETURN |
                  EXIT |
                  HELP [<cmd_keyword>]
                  IDS_ON.

<output_cmd> ::= SAY (<idsvar> | <string>) |
                WRITE (<idsvar> | <string>).

<regtest_cmd> ::= REGTESTSTART |
                  REGTESTEND

<modclass_cmd> ::= DCL_OBJ <objvar> <ident> [<parlist>] .|
                  DCL_REF_OBJ <idsvar> REF <ident> |
                  DISPOSE_OBJ objvar |
                  ASSIGN_REF_OBJ <objvar> <idsvar> |
                  ASSIGN_DEREF_PTR <idsvar> <objvar> |
                  SHOW_OBJ <objvar> |
                  OBJ_LIST (<ident> | ALL |
                  OBJ_REF_LIST (<ident>) | ALL) |
                  CLASS_LIST.

<assign_cmd> ::= <idsvar> "!=" (methcall_cmd) | <chars> |
                <objvar> "!=" (<objvar> | <methcall_cmd>).

```



7

```

<methcall_cmd> ::= <objvar> "." <ident> <parlist>.

<cmd_keyword> ::= DCL | END | ... | CLASS_LIST.

<param>..... ::= <objvar> | <idsvar>.

<ident>      ::= <letter> { <letter> | <number> | "_" }.

<idsvar>     ::= "%" {<letter> | <number> | <special>}+.

<objvar>     ::= "&" <letter> {<letter> | <number> | "_" }.

<string>     ::= "'" {<beliebiges Zeichen des EBCDIC-Codes
                        außer '>}' "'".

<chars>      ::= {<beliebiges Zeichen des EBCDIC-Codes
                        außer;>}.

<letter>     ::= "A" | "B" | ... | "Z".

<number>     ::= "O" | "1" | ... | "g".

<special>    ::= <Für IDS-Variablenamen einschließlich Kompo-
                        nenten erlaubte Sonderzeichen>.

```

Beispielsweise werden im erfindungsgemäßen Verfahren zwei Arten von Variablen unterschieden. Zum einen die reinen IDS-Variablen (IDS: Interaktives Debugging System), die wie gewohnt deklariert und benutzt werden können, und zum anderen Objekte, die als Instanzen von Testlings- oder Parameterklassen über das Kommando DCL\_OBJ deklariert werden.

Beispielsweise Variable aus mitgebundenen Moduln können über IDS und in Bool'schen Bedingungen von Schleifen und Verzweigungen oder auf der rechten Seite von Zuweisungen mit IDS verwendet werden. Die Verbindung zwischen den mit DCL\_OBJ vereinbarten



Objekten mit IDS wird durch IDS-Pointer beispielsweise vom Mode "REF Testingsklasse" (mit DCL\_REF\_OBJ zu vereinbaren) und den zwei Kommandos ASSIGN\_REF\_OBJ und ASSIGN\_DEREF\_PTR hergestellt. Beispielsweise werden Operatoren der Testsystem Kommandosprache auf IDS-Operatoren abgebildet.

Im erfindungsgemäßen Verfahren können beispielsweise folgende Befehle zu Objektbearbeitung vorgesehen sein, die vom Testenden interaktiv eingegeben werden und die eine hohe Flexibilität beim Test gewährleisten. Die Testeingaben durch den Testenden und die von ihm getätigten Aufrufe werden sinnvollerweise gesondert verwaltet. Beispielsweise ist eine Objektverwaltung vorgesehen, um die Zuweisung von Testobjekten in der Kommandosprache zu Rechner-Objekten sicherzustellen. Weiterhin kann auch vorteilhaft eine Aufrufbearbeitung vorgesehen sein, welche die Konsistenz von Test-Methodenaufrufen mit hardwareseitig ablaufenden Programmen sicherstellt. Es ist dazu nicht erforderlich, das Testprogramm neu zu generieren.

#### **Kommandos zur Objektbearbeitung**

##### **Vereinbarung eines Objekts einer MODULE-Klasse**

DCL\_OBJ Objektname Klassenname "("Parameterliste")"

Mit diesem Kommando wird ein Objekt auf Ebene der Kommandosprache bereitgestellt. Die Parameterliste gibt die Parameter für den Konstruktoraufruf an. Sie kann entfallen, dann wird der Standardkonstruktor aufgerufen. Ansonsten gelten für sie die gleichen Regeln wie beim Methodenaufruf.

Der Objektname muß aus dem Zeichen & gefolgt von einem Großbuchstaben und dann höchstens noch 29 Großbuchstaben, Ziffern oder Unterstrichen bestehen, ansonsten wird die Fehlermeldung "ILLEGAL OBJECTNAME" ausgegeben. Die Klasse muß bei der Erzeugung



des Testrahmens mit berücksichtigt worden sein, ansonsten wird die Fehlermeldung "UNKNOWN CLASS" ausgegeben. Falls schon ein Objekt mit dem angegebenen Namen vereinbart wurde, wird die Fehlermeldung "OBJECT ALREADY DECLARED" ausgegeben. Der Kommandoname DCL\_OBJ wurde zur Unterscheidung vom IDS-Kommando DCL gewählt.

### **Vereinbarung eines Pointers auf Objekte**

DCL\_REF\_OBJ Pointername REF Klassenname

Der vereinbarte Pointer ist eine IDS-Variable, von deren Existenz die Objektverwaltung Kenntnis hat. Die Regeln von IDS sind einzuhalten.

Der Pointermode auf Klassen, von denen Objekte vereinbart werden können, wird im Testsystem intern deklariert. Beim Löschen von Objekten mit dem DISPOSE\_OBJ-Kommando wird geprüft, ob noch mit DCL\_REF\_OBJ für die Klasse des zu löschenden Objekts vereinbarte Pointer auf dieses zeigen (vgl. DISPOSE\_OBJ-Kommando).

### **Löschen eines Objekts einer MODULE-Klasse**

DISPOSE\_OBJ Objektname

Mit diesem Kommando wird ein zuvor mit DCL\_OBJ instantiiertes Objekt wieder gelöscht. Das Objekt muß zuvor erzeugt worden sein, sonst liegt ein Fehlerfall vor, der mit "OBJECT UNDEFINED" gemeldet wird. Falls noch mit DCL\_REF\_OBJ für die Klasse des zu löschenden Objekts vereinbarte Pointer auf das Objekt zeigen, wird das Kommando nicht ausgeführt und die Fehlermeldung "OBJECT YET REFERENCED BY POINTER" ausgegeben. Es wird aber nicht geprüft, ob noch andere als mit DCL\_REF\_OBJ vereinbarte Pointer auf das Objekt existieren.



### **Übergabe der Adresse eines Objekts**

ASSIGN\_REF\_OBJ Objektname IDS-Variable

Das Kommando schreibt die Adresse des angegebenen Objekts in die IDS-Variable. Diese muß mit DCL\_REF\_OBJ als Pointer auf die Klasse des angegebenen Objekts vereinbart worden sein, sonst wird die Fehlermeldung "UNKNOWN IDS-POINTER" ausgegeben. (Kein Polymorphismus für die mit DCL\_REF\_OBJ Pointer!). Das Objekt muß zuvor mit DCL\_OBJ erzeugt worden sein, sonst wird die Fehlermeldung "UNKNOWN OBJECT" ausgegeben.

Nach dem Kommando zeigt die IDS-Variable auf das Objekt. Durch dieses Kommando wird für den Tester die Verbindung zwischen IDS und den Objekten der Kommandosprache hergestellt.

### **Übergabe der Daten eines dereferenzierten Pointers**

ASSIGN\_DEREF\_PTR IDS-Variable Objektname

Das Kommando kopiert die Daten des Objekts, auf das der Pointer zeigt, in das angegebene Objekt. Durch dieses Kommando wird für den Tester die Verbindung zwischen IDS und den Objekten der Kommandosprache hergestellt.

Die IDS-Variable muß mit DCL\_REF\_OBJ als Pointer auf die Klasse des angegebenen Objekts vereinbart worden sein, sonst wird die Fehlermeldung "UNKNOWN IDS-POINTER" ausgegeben. (Kein Polymorphismus für die mit DCL\_REF\_OBJ vereinbarten!). Die IDS-Variable muß auch tatsächlich auf ein Objekt dieser Klasse zeigen. Dies kann vom Testsystem nicht überprüft werden! Das angegebene Objekt muß zuvor mit DCL\_OBJ erzeugt worden sein, sonst wird die Fehlermeldung "UNKNOWN OBJECT" ausgegeben.

### **Ausgabe der Daten eines Objekts**



### SHOW\_OBJ Objektname

Das Kommando veranlaßt die Ausgabe der Datenattribute des angegebenen Objekts in die Protokolldaten ALL, OUT und auf Terminal, falls sie aktiviert sind (dazu Kommando PROTSTATUS). Es wird mit dem IDS-Kommando DISPLAY realisiert. Das Objekt muß zuvor mit DCL\_OBJ erzeugt worden sein, ansonsten wird die Fehlermeldung "UNKNOWN OBJECT" ausgegeben.

Eine weitere Möglichkeit, Objektdaten auszugeben, ist das Anzeigen dereferenziert IDS-Pointer, die auf das Objekt zeigen, mittels der Kommandos SAY und WRITE.

### **Aufruf gewöhnlicher Methoden**

Objektname.Methodenname "("Parameterliste")"

Bei dem angegebenen Objekt wird die Methode mit den Parametern der Parameterliste aufgerufen. Es muß zuvor mit DCL\_OBJ erzeugt worden sein. Als Parameter sind nur IDS-Variablen oder Objekte (die zuvor in der Kommandosprache vereinbart wurden) zugelassen. Bei fehlerhaften Modes wird die Fehlermeldung "PARAMETERMODES DO NOT FIT TO METHODNAME" ausgegeben. Wegen möglichen Overloadings in den Methoden der Testlingsklasse kann der Mode-Fehler im allgemeinen nicht genauer lokalisiert werden. Die Parameter sind durch Kommata zu trennen, derart, daß zwischen zwei direkt aufeinanderfolgenden Parametern je genau ein Komma steht.

od

### **Aufruf einer Methode mit Ergebnis**

Variable := Objektname.Methodenname "("Parameterliste")"



Es gelten die gleichen Regeln wie beim Aufruf gewöhnlicher Methoden (s.o.).

Die Variable, der das Ergebnis zugewiesen wird, muß eine IDS-Variable oder ein zuvor mit DCL\_OBJ vereinbartes Objekt passenden Modes sein, sonst wird eine Fehlermeldung ausgegeben.

#### **Zuweisung mit Objekten**

Obj1 := Obj2

Den Datenkomponenten von Obj1 werden die entsprechenden Werte von Obj2 zugewiesen. Die beiden Objekte müssen zuvor mit DVL\_OBJ vereinbart worden und von der selben Klasse sein, sonst wird die Fehlermeldung "UNKNOWN OBJECT Objektname" bzw. "MODE-MISSMATCH" ausgegeben.

#### **Information über Objekte**

OBJ\_LIST Klassenname

Dieses Kommando veranlaßt die Ausgabe der Namen aller Objekte, die aktuell für die angegebene Klasse vereinbart sind. Statt Klassenname kann auch ALL angegeben werden, worauf zu sämtlichen Klassen, für die eine Objektverwaltung existiert, die Objektnamen ausgegeben werden. Falls für die Klasse mit dem angegebenen Namen keine Objektverwaltung generiert wurde, wird die Fehlermeldung "UNKNOWN CLASS" ausgegeben..

#### **Information über Objektpointer**

OBJ\_REF\_LIST Klassenname

Dieses Kommando veranlaßt die Ausgabe aller Pointernamen, die aktuell zur angegebenen Klasse mit dem Kommando DCL\_REF\_OBJ



vereinbart sind. Statt Klassennamen kann auch ALL angegeben werden, worauf sämtliche mit DCL REF OBJ vereinbarten Objektpointernamen ausgegeben werden. Falls für die Klasse mit dem angegebenen Namen keine Objektverwaltung generiert wurde, wird die Fehlermeldung "UNKNOWN CLASS" ausgegeben.

### Information über Klassen

#### CLASS\_LIST

Dieses Kommando veranlaßt die Ausgabe aller Klassennamen, für die eine Objektverwaltung generiert wurde.

Die hier gemachten Ausführungen sind nur als Beispiel für eine Ausführungsform des erfindungsgemäßen Verfahrens aufzufassen und berücksichtigen die spezielle Namensgebung von Object-CHILL. Sie sind auf andere objektorientierte Programmiersprachen wie zum Beispiel C++, Eiffel, SIMULA oder Smalltalk direkt übertragbar. Es kann beispielsweise ohne Beschränkung der Allgemeinheit auch vorgesehen sein, andere Kommandos für einen Testkommando-Interpreter vorzusehen. Besonders wichtig ist, daß nur einmal ein Testprogramm übersetzt werden muß und daß der Tester durch das erfindungsgemäße Verfahren die Möglichkeit erhält, sämtliche für die Klassentests erforderliche Testfunktionen interaktiv auszulösen. Interaktiv kann in diesem Zusammenhang auch bedeuten, daß er Testkommandodateien generiert und diese startet und mit entsprechenden Abbruchkriterien versieht.

In Figur 2 ist ein Testbeispiel des erfindungsgemäßen Verfahrens als Programmlisting aufgeführt. Es enthält die Klassenvereinbarung und die entsprechenden Kommandos für den Testinterpreter.

In diesem Beispiel wird die generische Klasse STAGENLS über ihre zwei Ausprägungen STAINTLS und STATEXLS getestet.



Für den Generator G dient eine Datei als Eingabe, die drei Klassenspezifikationen Member 1), 2) und 3) wie gezeigt als Inhalt hat (die Reihenfolge der Members in der Datei ist unerheblich!).

Die Klasse STATEXLS importiert aus einem CHILL-Modul TEXMODE. Der exportierende Modul ist keine Eingabe für den Generatorlauf.

Bei diesem Beispiel werden zwei Objektverwaltungsklassen OMCL001S und OMCL002S und zwei Aufrufbearbeitungsklassen MCCL001S und MCCL002S generiert, weil nur zwei Klassen auftreten, von denen Objekte instantiiert und dort Methoden aufgerufen werden können, nämlich STAINTLS und STATEXLS. Bei der oben angegebenen Reihenfolge der Memberdateien gehören die Klassen ... 001S zu STAINTLS und die ... 002S zu STATEXLS.

Figur 3 zeigt anhand eines Blockdiagrammes das Zusammenwirken verschiedener Komponenten, bei der Durchführung eines beispielhaft ausgebildeten erfindungsgemäßen Verfahrens.

An oberster Stelle befindet sich hier der Kommandointerpreter KI, welcher Eingaben einer Testperson interpretiert. Die Objektverwaltung OV und die Aufrufverwaltung AV stellen die Konsistenz von Testlingen T mit dem Testszenario eines Testenden sicher. Dies geschieht, indem sie beispielsweise Speicherbereiche im Rechner über Pointer zuweisen, oder diesen Speicherbereichen namen zuordnen. So gewährleisten sie den sicheren Ablauf der auszuführenden Methoden.



## 4 EINGABEN

### 4.1 Die Kommandosprache des Testsys

#### 4.1.1 Allgemeine Konventionen

Die Kommandosprache des Testsystems ist an CHILL und an IDS angelehnt. Dies bedeutet insbesondere, daß

- der gleiche Zeichensatz verwendet wird, d.h. für Schlüsselwörter und Bezeichner nur Großbuchstaben zulässig sind,
- alle Kommandos durch ein Semikolon abgeschlossen werden,
- Kommentare wie Leerzeichen (Blanks) behandelt werden,
- Blanks zwischen Schlüsselwörtern oder Bezeichnern angegeben werden müssen, während sie zwischen Operatoren wegbleiben können.

Zusätzlich werden folgende Festlegungen getroffen:

- Zeilentrenner werden wie Blanks behandelt, d.h. bei einem mehrzeiligen Kommando wird nach dem 80. Zeichen (entspricht einer MVS-Zeile) ein Blank eingefügt. Somit endet ein Wort spätestens am Zeilenende.
- Die im folgenden aufgeführten Längen verstehen sich als Maximalwerte.  
Bezeichner sind maximal 31 Zeichen lang.  
Strings können eine Zeile (80 Zeichen) lang sein, wenn sie mit den Kommandos SAY bzw. WRITE ausgegeben werden.  
Boolesche Bedingungen in Schleifen und Verzweigungen, Ausdrücke und Variablendeklarationen dürfen 190 Zeichen nicht überschreiten, da diese mit Hilfe von IDS ausgewertet bzw. deklariert werden.
- Erfolgreich eingegebene Kommandos werden nicht explizit quittiert, sondern es wird die Eingabe des nächsten Kommandos erwartet. Ausnahmen hierzu bilden lediglich die Kommandos END, REGTESTSTART und REGTESTEND.
- Fehlerhafte Kommandos werden durch entsprechende Meldungen angezeigt, wobei implizite Fehlerkorrekturen nicht durchgeführt werden. Nach einem fehlerhaften Kommando erwartet der Interpreter das nächste Kommando im Terminalmodus, d.h. Kommandodateien werden unmittelbar nach einem Fehler verlassen. Tritt der Fehler in Kontrollstrukturen auf, werden diese genau wie das Kommando abgebrochen. Weitere Kommandos in der gleichen Zeile werden ignoriert. Fehlerhafte Kommandos während des Regressionstests führen zum Abbruch des Regressionstests. (vgl. Kapitel 6)



#### 4.1.2 Zur Beschreibung der Kommandosprache

Die Beschreibung der Kommandos ist an die erweiterte Backus-Naur-Form (EBNF) angelehnt, dabei gelten folgende Regeln:

- Schlüsselwörter der Kommandosprache werden in Großbuchstaben geschrieben.
- Alternativen werden durch '|' dargestellt, z.B.:  
Com hat folgende Form: FORM1 | FORM2 , d.h. das Kommando Com kann in der Form FORM1 oder FORM2 eingegeben werden.
- Teile, die weggelassen werden können, werden in '[' und ']' geklammert.  
Z.B.: Com hat folgende Form: TEIL1 [ TEIL2] , d.h. das Kommando Com kann in der Form TEIL1 oder TEIL1 TEIL2 dargestellt werden.
- Zusammengehörige Teile werden in '(' und ')' geklammert.  
Z.B.: Com hat folgende Form: (TEIL1 | TEIL2) TEIL3, d.h. das Kommando Com kann in der Form TEIL1 TEIL3 oder TEIL2 TEIL3 dargestellt werden.
- Bei Kommandos oder Kommandoparametern, die abgekürzt werden können, ist die Abkürzung der unterstrichene Teil des Namens.  
Z.B. kann KURZFORM durch KURF abgekürzt werden.

#### 4.1.3 Die Syntax im Überblick

Die Syntax der Kommandosprache ist ebenso wie die einzelnen Kommandos in den folgenden Kapiteln mittels EBNF beschrieben. Zusätzlich werden folgende Festlegungen getroffen:

- Non-Terminal-Symbole, die dazu dienen, die Grammatik strukturiert gliedern zu können, werden in spitze Klammern ('<' und '>') gefaßt.
- Terminal-Symbole sind Symbole, die zur TeS-Kommandosprache gehören. Die Schlüsselwörter werden groß geschrieben und sonstige Symbole in Anführungszeichen (z.B. ":=") eingeschlossen
- Die links vom Metazeichen '::=' stehenden Non-Terminal-Symbole können durch die rechte Seite ersetzt werden. Die Grammatik ist kontextfrei, d.h. auf der linken Seite stehen keine zusätzlichen Terminal-Symbole.
- Symbole, die beliebig oft (auch 0-mal) wiederholt werden dürfen, werden in '(' und ')' geklammert. Folgt der schließenden Klammer ein '+', so bedeutet dies, daß das Symbol mindestens einmal wiederholt werden muß. Folgt eine Zahl x, so muß der geklammerte Teil genau x-mal wiederholt werden.



- Das Metasymbol '.' zeigt das Ende einer Regel an.
- Anstelle des Non-Terminal-Symbols <chars> wird eine beliebige Zeichenfolge erwartet, die zur Auswertung an IDS übergeben wird. Diese Zeichenfolge wird vom Testsystem nicht syntaktisch analysiert.

#### Grammatik der Kommandosprache für den MODULE-Klassen-Test:

```

<statements> ::= {<command> ";"}.

<command> ::= <control_cmd> | <general_cmd> | <regtest_cmd> |
               <output_cmd> | <modclss_cmd> | <assign_cmd> |
               <methcall_cmd>

<control_cmd> ::= DO WHILE <chars> ";" <statements> OD |
                  IF <chars> THEN <statements>
                      [ELSE <statements>] FI.

<general_cmd> ::= DCL <chars> |
                  END |
                  PROTSTATUS |
                  PROTOCOL [TERM] [ALL] [INP] [OUT] |
                  INPUT (INP1 | INP2 | INP3 | INP4 | INP5) |
                  RETURN |
                  EXIT |
                  HELP [<cmd_keyword>] |
                  IDS_ON.

<output_cmd> ::= SAY (<idsvar> | <string>) |
                  WRITE (<idsvar> | <string>).

<regtest_cmd> ::= REGTESTSTART |
                  REGTESTEND.

<modclss_cmd> ::= DCL_OBJ <objvar> <ident> [<parlist>] |
                  DCL_REF_OBJ <idsvar> REF <ident> |
                  DISPOSE_OBJ <objvar> |
                  ASSIGN_REF_OBJ <objvar> <idsvar> |
                  ASSIGN_DEREF_PTR <idsvar> <objvar> |
                  SHOW_OBJ <objvar> |
                  OBJ_LIST (<ident> | ALL) |
                  OBJ_REF_LIST (<ident> | ALL) |
                  CLASS_LIST.

<assign_cmd> ::= <idsvar> ":=" (<methcall_cmd> | <chars>) |
                  <objvar> ":=" (<objvar> | <methcall_cmd>).

<methcall_cmd> ::= <objvar> "." <ident> <parlist>.

<cmd_keyword> ::= DCL | END | ... | CLASS_LIST.

<parlist> ::= "(" [ <param> ("," <param>) ] ")".

<param> ::= <objvar> | <idsvar>.

```



## Patentansprüche

1. Verfahren zum Testen mindestens einer Klasse eines objektorientierten Programmes auf einem Rechner,

- a) bei dem aus mindestens einer Klassenspezifikation eine Methodeninformation über mindestens eine Methode und deren zugehörige Aufrufparameter gewonnen wird,
- b) bei dem mit Hilfe der Methodeninformation ein Bestandteil eines Testprogramms erzeugt wird, welcher mindestens eine Zuweisung von beim Testen erforderlichen Rechnerbetriebsmitteln sicherstellt,
- c) bei dem ein Testprogramm erzeugt wird, das als einen Bestandteil einen Kommandointerpreter enthält, welcher von einem Testenden eingegebene Testkommandos interpretiert, und das als einen weiteren Bestandteil den unter b) erzeugten Bestandteil des Testprogramms enthält
- d) und bei dem das Testprogramm mit Hilfe der Testkommandos zum Testen der Klasse verwendet wird.

2. Verfahren nach Anspruch 1, bei dem die Methodeninformation zumindest für alle über eine externe Methodenschnittstelle exportierten Methoden, jeder einzelnen von gemeinsam zu testenden Klassen gewonnen wird.

3. Verfahren nach einem der Ansprüche 1 oder 2, bei dem ein Rechnerbetriebsmittel Speicher ist.

4. Verfahren nach einem der Ansprüche 1 bis 3, bei dem die Zuweisung der Betriebsmittel über Pointer erfolgt.

5. Verfahren nach Anspruch 1, bei dem der Kommandointerpreter mindestens einen in einem Betriebssystem des Rechners vorhandenen Debugger benutzt.



6. Verfahren nach einem der Ansprüche 1 bis 5, bei dem mindestens eine interne Datenkomponente von mindestens einem aus einer Klasse abgeleiteten Objekt gelesen und/oder eingeschrieben wird.

7. Verfahren nach einem der Ansprüche 1 bis 6, bei dem die Methodeninformation für mindestens eine Klasse mit mindestens einer ererbten Methode, auch aus mindestens einer vererbenden Klasse gewonnen wird.

8. Verfahren nach einem der Ansprüche 1 bis 7 für mindestens eine aus einer generischen Klasse abgeleitete Klasse, bei dem die Methodeninformation auch für mindestens eine Methode aus der generischen Klasse gewonnen wird.



1 / 2

FIG 1

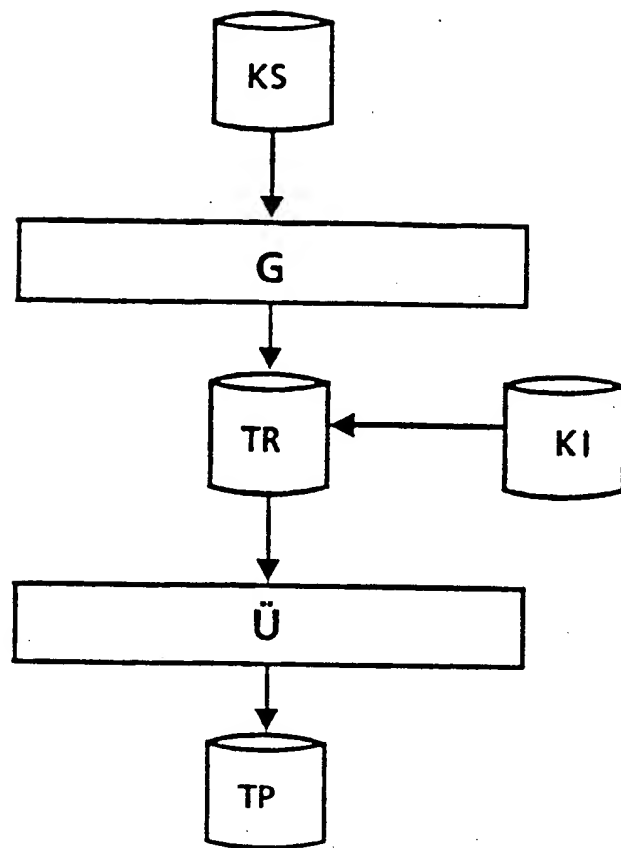
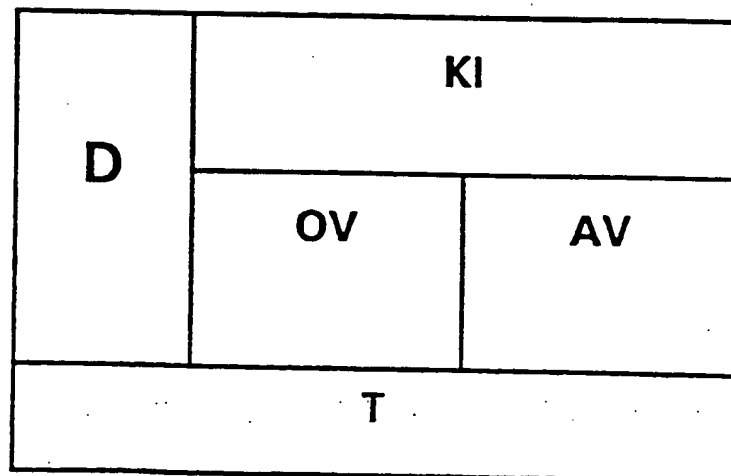


FIG 3





2/2

## FIG 2

Member 1)

```
STAGENLS : GENERIC
SYNMODE ELEMMODE = ANY_ASSIGN;
MODULE CLASS
GRANT POP, TOP, ISEMPY, PUSH PREFIXED STAGENLS;
STAGENLS : CONSTR(); END STAGENLS;
```

```
POP : PROC();
END POP;
```

```
TOP : PROC() RETURNS(ELEMMODE);
END TOP;
```

```
PUSH : PROC(EL ELEMMODE);
END PUSH;
```

```
ISEMPY : PROC() RETURNS(BOOL);
END ISEMPY;
```

```
SYNMODE STACKPTRMODE = REF STACKELMODE;
SYNMODE STACKELMODE = STRUCT(
    EL ELEMMODE,
    NEXT STACKPTRMODE);
DCL ANCHOR STACKPTRMODE;
```

```
END STAGENLS;
```

Der BODY von STAGENLS ist nicht Eingabe für den Teststrahmengenerator!

Member 2)

```
STAINTLS : MODULE CLASS = STAGENLS
ACT_GEN_PART
SYNMODE ELEMMODE = INT;
END STAINTLS;
```

Member 3)

```
STATEXLS : MODULE CLASS = STAGENLS
SEIZE TEXMODE;
ACT_GEN_PART
SYNMODE ELEMMODE = TEXMODE;
END STATEXLS;
```



## INTERNATIONAL SEARCH REPORT

Int. onal Application No.

PCT/EP 93/03450

A. CLASSIFICATION OF SUBJECT MATTER IPC 5 G06F11/00		
According to International Patent Classification (IPC) or to both national classification and IPC		
B. FIELDS SEARCHED		
Minimum documentation searched (classification system followed by classification symbols) IPC 5 G06F		
Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched		
Electronic data base consulted during the international search (name of data base and, where practical, search terms used)		
C. DOCUMENTS CONSIDERED TO BE RELEVANT		
Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	AUTOMATISIERUNGSTECHNISCHE PRAXIS - ATP vol. 31, no. 1, January 1989, MUNCHEN DE pages 30 - 36 G. PAUTHNER ET AL. 'TBM, ein vollständig generierbares Software-Modultestbett' see page 32, left column, line 1 - right column, line 15 ---	1-8
Y	WO,A,90 04829 (EASTMAN KODAK COMPANY) 3 May 1990 see page 49, line 1 - line 16 --- -/--	1-8
<input checked="" type="checkbox"/> Further documents are listed in the continuation of box C. <input checked="" type="checkbox"/> Patent family members are listed in annex.		
* Special categories of cited documents : "A" document defining the general state of the art which is not considered to be of particular relevance "E" earlier document but published on or after the international filing date "L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified) "O" document referring to an oral disclosure, use, exhibition or other means "P" document published prior to the international filing date but later than the priority date claimed "T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention "X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone "Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art. "&" document member of the same patent family		
Date of the actual completion of the international search 28 April 1994		Date of mailing of the international search report 06.05.94
Name and mailing address of the ISA European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax (+31-70) 340-3016		Authorized officer Corremans, G



## INTERNATIONAL SEARCH REPORT

Int. Patent Application No.

PCT/EP 93/03450

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT		
Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	PROCEEDINGS OF THE CONFERENCE ON SOFTWARE MAINTENANCE-1988 27 October 1988 , PHOENIX, ARIZONA pages 404 - 408 TOMAZ DOGSA ET AL. 'CAMOTE - Computer aided module testing and design environment' see page 406, left column, line 1 - line 41 ---	1-8
A	PROCEEDINGS OF THE 9TH DIGITAL AVIONICS SYSTEMS CONFERENCE 18 October 1990 , VIRGINIA BEACH, VIRGINIA pages 107 - 110 GARY L. DEHLIN 'Automating Test Driver Generation' see page 108, left column, line 22 - line 44 -----	1-8



# INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No.

PCT/EP 93/03450

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO-A-9004829	03-05-90	US-A- 4989132	29-01-91
		EP-A- 0439533	07-08-91
		JP-T- 4501477	12-03-92
-----			



A. KLASSIFIZIERUNG DES ANMELDUNGSGEGENSTANDES  
IPK 5 G06F11/00

Nach der Internationalen Patentklassifikation (IPK) oder nach der nationalen Klassifikation und der IPK

B. RECHERCHIERTE GEBIETE

Recherchierte Mindestprüfstoff (Klassifikationssystem und Klassifikationsymbole)  
IPK 5 G06F

Recherchierte aber nicht zum Mindestprüfstoff gehörende Veröffentlichungen, soweit diese unter die recherchierten Gebiete fallen

Während der internationalen Recherche konsultierte elektronische Datenbank (Name der Datenbank und evtl. verwendete Suchbegriffe)

C. ALS WESENTLICH ANGESEHENE UNTERLAGEN

Kategorie	Bezeichnung der Veröffentlichung, soweit erforderlich unter Angabe der in Betracht kommenden Teile	Betr. Anspruch Nr.
Y	AUTOMATISIERUNGSTECHNISCHE PRAXIS - ATP Bd. 31, Nr. 1, Januar 1989, MUNCHEN DE Seiten 30 - 36 G. PAUTHNER ET AL. 'TBM, ein vollständig generierbares Software-Modultestbett' siehe Seite 32, linke Spalte, Zeile 1 - rechte Spalte, Zeile 15 ---	1-8
Y	WO, A, 90 04829 (EASTMAN KODAK COMPANY) 3. Mai 1990 siehe Seite 49, Zeile 1 - Zeile 16 --- -/--	1-8

☒ Weitere Veröffentlichungen sind der Fortsetzung von Feld C zu entnehmen

☒ Siehe Anhang Patentfamilie

\* Besondere Kategorien von angegebenen Veröffentlichungen :

\*A\* Veröffentlichung, die den allgemeinen Stand der Technik definiert, aber nicht als besonders bedeutsam anzusehen ist

\*E\* älteres Dokument, das jedoch erst am oder nach dem internationalen Anmeldedatum veröffentlicht worden ist

\*L\* Veröffentlichung, die geeignet ist, einen Prioritätsanspruch zweifelhaft erscheinen zu lassen, oder durch die das Veröffentlichungsdatum einer anderen im Recherchenbericht genannten Veröffentlichung belegt werden soll oder die aus einem anderen besonderen Grund angegeben ist (wie ausgeführt)

\*O\* Veröffentlichung, die sich auf eine mündliche Offenbarung, eine Benutzung, eine Ausstellung oder andere Maßnahmen bezieht

\*P\* Veröffentlichung, die vor dem internationalen Anmeldedatum, aber nach dem beanspruchten Prioritätsdatum veröffentlicht worden ist

\*T\* Spätere Veröffentlichung, die nach dem internationalen Anmeldedatum oder dem Prioritätsdatum veröffentlicht worden ist und mit der Anmeldung nicht kollidiert, sondern nur zum Verständnis des der Erfindung zugrundeliegenden Prinzips oder der ihr zugrundeliegenden Theorie angegeben ist

\*X\* Veröffentlichung von besonderer Bedeutung, die beanspruchte Erfindung kann allein aufgrund dieser Veröffentlichung nicht als neu oder auf erfinderischer Tätigkeit beruhend betrachtet werden

\*Y\* Veröffentlichung von besonderer Bedeutung, die beanspruchte Erfindung kann nicht als auf erfinderischer Tätigkeit beruhend betrachtet werden, wenn die Veröffentlichung mit einer oder mehreren anderen Veröffentlichungen dieser Kategorie in Verbindung gebracht wird und diese Verbindung für einen Fachmann naheliegend ist

\*Z\* Veröffentlichung, die Mitglied derselben Patentfamilie ist

Datum des Abschlusses der internationalen Recherche

28. April 1994

Absendedatum des internationalen Recherchenberichts

06.05.94

Name und Postanschrift der Internationale Recherchenbehörde  
Europäisches Patentamt, P.B. 5818 Patentlaan 2  
NL - 2280 HV Rijswijk  
Tel. (+31-70) 340-2040, Tx. 31 651 epo nl,  
Fax (+31-70) 340-3016

Bevollmächtigter Bediensteter

Corremans, G



# INTERNATIONALER RECHERCHENBERICHT

Internationales Aktenzeichen:

PCT/EP 93/03450

C.(Fortsetzung) ALS WESENTLICH ANGESEHENE UNTERLAGEN

Kategorie	Bezeichnung der Veröffentlichung, soweit erforderlich unter Angabe der in Betracht kommenden Teile	Betr. Anspruch Nr.
-----------	--	--------------------

A	<p>PROCEEDINGS OF THE CONFERENCE ON SOFTWARE MAINTENANCE-1988 27. Oktober 1988 , PHOENIX, ARIZONA Seiten 404 - 408 TOMAZ DOGSA ET AL. 'CAMOTE - Computer aided module testing and design environment' siehe Seite 406, linke Spalte, Zeile 1 - Zeile 41</p>	1-8
---	---	-----

A	<p>PROCEEDINGS OF THE 9TH DIGITAL AVIONICS SYSTEMS CONFERENCE 18. Oktober 1990 , VIRGINIA BEACH, VIRGINIA Seiten 107 - 110 GARY L. DEHLIN 'Automating Test Driver Generation' siehe Seite 108, linke Spalte, Zeile 22 - Zeile 44</p>	1-8
---	--	-----